Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	12 mar 2013
Sal	TER1
Tid	14-18
Kurskod	TDDB68
Provkod	TEN1
Kursnamn/benämning	Processprogrammering och operativsystem
Institution	IDA
Antal uppgifter som ingår i tentamen	8
Antal sidor på tentamen (inkl. försättsbladet)	6
Jour	Christoph Kessler, 0703-666687
Besöker salen ca kl.	16:00
Examinator/kursansvarig	Christoph Kessler, IDA
Kursadministratör (namn + tfnnr + mailadress)	Carita Lilja, 013-281463, carita.lilja@liu.se
Tillåtna hjälpmedel	Engelsk ordbok, miniräknare
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	

Linköpings universitet
IDA Department of Computer and Information Sciences
Prof. Dr. Christoph Kessler

TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem / *Concurrent programming and operating systems*

12 mar 2013, 14:00–18:00 TER1

Jour: Christoph Kessler (070-3666687, 013-282406), visiting ca. 16:00

Hjälpmedel / *Admitted material:*

– Engelsk ordbok / *Dictionary from English to your native language*;
– Miniräknare / *Pocket calculator*

General instructions

- This exam has 8 assignments and 5 pages, including this one.
  Read all assignments carefully and completely before you begin.
- It is recommended that you use a new sheet of paper for each assignment, because they will be corrected by different persons.
  Sort the pages by assignment, number them consecutively and mark each one on top with your exam ID and the course code.
- You may answer in either English or Swedish. **English is preferred** because not all correcting assistants understand Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- Motivate clearly all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are *not* ordered according to difficulty.
- The exam is designed for 40 points. You may thus plan about 5 minutes per point.
- Grading: U, 3, 4, 5. The preliminary threshold for passing is 20 points.

1. (6.5 p.) **Interrupts, processes and threads**

   (a) Define the terms *process*, *kernel-level thread* and *user-level thread*, and explain the differences between them. (3p)

   (b) The *system call API* (application programming interface) is a software abstraction for invoking OS functionality.

   Name one kind of technical details that it abstracts from. (0.5p)

   Why is such abstraction important for application programming? (0.5p)

   (c) Two main methods for inter-process communication in a computer are *shared memory* and *message passing*.

   For each of them, give a short explanation of how it works and how the operating system is involved, i.e., which important system calls are to be used and what they do.

   Which of the two methods is likely to have less overhead if two processes communicate frequently with each other, and why? (2.5p)

2. (5 p.) **CPU Scheduling**

   Given a single-CPU system and the following set of processes with arrival times (in milliseconds), expected maximum execution time (ms), and priority (1 is highest, 5 is lowest priority).

Process	Arrival time	Execution time	Priority (as applicable)
P_1	0	7	5
P_2	2	5	3
P_3	6	2	4
P_4	7	3	2
P_5	11	1	1

   For each of the following scheduling algorithms, create a Gantt chart (time bar diagram, starting at $t = 0$) that shows when the processes will execute on the CPU. Where applicable, the time quantum will be 3 ms. Assume that a task will be eligible for scheduling immediately on arrival. If you need to make further assumptions, state them carefully and explain your solution. (5p)

   (i) FIFO;

   (ii) Round-robin;

   (iii) Shortest Job First *without* preemption;

   (iv) Priority Scheduling *without* preemption.

   (v) Priority Scheduling *with* preemption.

3. (7 p.) **Synchronization**

Money transfer between bank accounts

A bank administrates N accounts, which are, for simplicity, numbered 0 to $N-1$ and stored in an array:

```
struct {
 float balance;    // current balance value
 unsigned int owner;
 // ...
} account[N];
```

For a transfer of amount $x > 0$ from account i to account j (where $i \neq j$), the bank software uses the following routine:

```
void transfer( float x, unsigned int i, unsigned int j )
{
  if (account[i].balance >= x) { // avoid negative balance
     account[i].balance -= x;
     account[j].balance += x;
  }
  else errmsg("Transfer denied: insufficient balance");
}
```

Initially, this bank program was executed as a single-threaded process that was processing a batch of transfer requests one at a time, calling the `transfer` function above.

As the number of accounts and the frequency of transfering money grew considerably, the bank decided to migrate its software to a multiprocessor system, storing the accounts in shared memory and allowing multiple threads to process transfer requests concurrently. It is your task to make the software thread-safe.

(a) Show with a concrete example scenario that, without proper synchronization, the above code contains a race condition that can lead to wrong results. (1p)

(b) Identify the critical section(s) and write a simple, thread-safe version of the `transfer` function above, using an *appropriate* mechanism. Explain all operations carefully. (2p)

(c) For high transfer request rates, the simple synchronized solution of the previous question may not scale, even if there are enough processors available. Why? (0.5p)

Suggest a more advanced protection scheme that allows many transfers to proceed simultaneously. Show the (pseudo)code and explain all operations. (1.5p)

May your new solution lead to deadlocks? If yes, state under what condition(s) and suggest a suitable mechanism for deadlock prevention. If no, explain why your solution is deadlock-free. (1p)

(d) What would be a more appropriate abstraction (programming construct) for this kind of operation, and why? (Give the technical term and a short explanation). (1p)

4. (5 p.) **Deadlocks**

(a) There are four conditions that must hold for a deadlock to become possible. Name and describe them briefly. (2p)

(b) You are given a system with 4 types of resources, A, B, C and D. There are 4 instances of A, 5 instances of B, 1 instance of C and 7 instances of D. Currently, 5 processes $P_1 ... P_5$ are running, and for each process, the resources currently held and its total maximum resource need (including the already held ones) for each type are given as follows:

Process	Already held	Maximum total need
	$A\ B\ C\ D$	$A\ B\ C\ D$
P_1	0 1 0 1	0 2 0 4
P_2	0 0 0 0	4 3 0 5
P_3	1 2 1 0	2 2 1 3
P_4	1 0 0 2	2 2 0 4
P_5	1 0 0 2	2 0 1 4

I.e., currently, process P_1 holds already one B and one D out of the 2 Bs and 4 Ds that it eventually may need in the worst case, etc. One A is currently still available, etc.

(i) Show that the system is currently in a safe state (calculation). (1.5p)

(ii) In the situation given above, process P_1 now asks for 1 instance of D, in addition to the B and D that it already has. Is it safe to grant the request? Why or why not? (calculation) (1.5p)

5. (7.5 p.) **Memory management**

(a) Consider a page-based virtual memory system with a page size of 512 bytes where virtual memory addresses have 32 bit size. If using *multi-level paging*,

i. determine how many levels of paging are required, and describe the structure of the virtual addresses (purpose, position and size of its bit fields); (1p)

ii. explain (annotated figure) how in this case the physical address is calculated by multi-level paging from a virtual address; (1p)

iii. show how a TLB can be used to accelerate address calculation; (1p)

iv. calculate the expected time for a paged memory access if a physical memory access costs 100ns on average and a TLB access costs 5ns, and the TLB hit rate is 80%. (0.5p)

(b) Explain how segmented virtual memory supports sharing of memory segments between processes. (1p)

(c) Characterize an important property of pages that are particularly suitable to choose as a victim when freeing a used frame in order to reduce page replacement overhead. Explain why. What kind of data structure could help with this choice? (1p)

(d) What is *thrashing* in a virtual memory system? How does it occur? And what can be done about it? (2p)

6. (4.5 p.) **File systems**

   (a) What information is usually contained in a *file control block* (FCB)? (At least 4 different items are expected) (1p)

   (b) Where is the FCB contents stored after a file has been opened? (0.5p)

   (c) Describe one technique to extend indexed allocation for large files. (1p)

   (d) Describe one case where the file system is *not* an appropriate abstraction for secondary storage, and explain why. (1p)

   (e) What is the purpose of disk scheduling? (1p)

7. (1.5 p.) **OS Structures and Virtualization**

   What does a *hypervisor* (also known as *virtual machine monitor, VM implementation*) do?

   Illustrate your answer with a commented figure that shows where the hypervisor is positioned in the system software stack and with which other system entities it interacts. (1.5p)

8. (3 p.) **Protection and Security**

   (a) Name two different (software or hardware) measures to prevent *buffer-overflow* vulnerabilities. Explain briefly (1 line each) how they work. (1p)

   (b) How does memory segmentation support protection? (1p)

   (c) Why can, in general, the *microkernel* approach increase the security of a system compared to a traditional OS structure? (1p)

Good luck!